# *DAY 3 OF MARKET_PLACE_HACKATHON*

## *API Integration and Data Migration Report*

**Prepared By: Rimsha Sheikh**

# Introduction:

Hackathons are fast-paced events where developers build innovative solutions within a short time frame. The day 3 of hackathon focuses on APIs and Sanity.io, leveraging their benefits such as scalability, real-time data handling, and seamless integration. APIs enable efficient data communication, while Sanity, as a headless CMS, empowers developers to create dynamic content structures. Together, they provide tools to build powerful, user-centric applications while reducing development time and effort.

# API Workflow:

## 1. Objective

To integrate APIs and migrate data into Sanity CMS to build a functional marketplace backend. This process includes utilizing APIs for reference, migrating data, and ensuring compatibility with schemas to replicate real-world practices.

# 1.Migration File:

```
import axios from 'axios';
import { client } from './sanityClient.js';

async function uploadImageToSanity(imageUrl: string): Promise<string> {

  try {
    // Fetch the image from the URL and convert it to a buffer
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);

    // Upload the image to Sanity
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop(), // Extract the filename from URL
    });

    // Debugging: Log the asset returned by Sanity
    console.log('Image uploaded successfully:', asset);

    return asset._id; // Return the uploaded image asset reference ID
  } catch (error) {
    console.error('❌ Failed to upload image:', imageUrl, error);
    throw error;
  }
}

async function importData() {
    try {
      // Fetch data from external API
      const response = await axios.get('https://fakestoreapi.com/products');
      const products = response.data;

      // Iterate over the products
      for (const product of products) {
        let imageRef = '';

        // Upload image and get asset reference if it exists
        if (product.image) {
          imageRef = await uploadImageToSanity(product.image);
        }

        const sanityProduct = {
          _id: `product-${product.id}`, // Prefix the ID to ensure validity
          _type: 'product',
          name: product.title,
          price: product.price,
          discountPercentage: product.discountPercentage || 0,
          tags: product.category ? [product.category] : [],
          image: {
            _type: 'image',
            asset: {
              _type: 'reference',
              _ref: imageRef, // Set the correct asset reference ID
            },
          },
          description: product.description,
          rating: product.rating?.rate || 0,
          ratingCount: product.rating?.count || 0,
        };

        // Log the product before attempting to upload it to Sanity
        console.log('Uploading product:', sanityProduct);

        // Import data into Sanity
        await client.createOrReplace(sanityProduct);
        console.log(`✅ Imported product: ${sanityProduct.name}`);
      }

      console.log('✅ Data import completed!');
    } catch (error) {
      console.error('❌ Error importing data:', error);
    }
}

importData();
```

The above data migration script utilizes the **Sanity Client** and makes use of a **fetch API** to import product data into the Sanity CMS system. Here's a breakdown of the functionality

## Sanity Client Initialization:

The `createClient` function is used to create a connection with your Sanity project. The `projectId` and `dataset` define which specific Sanity project and dataset will be used. The `apiVersion` and `token` are crucial for accessing your project securely.

## Image Upload Function

The `uploadImageToSanity` function handles image uploads. It fetches the image from the URL, converts it into a buffer, and then uploads it to Sanity CMS. This allows for seamless image management within the Sanity platform, associating images with the respective product data.

## Product Upload Function:

The `uploadProduct` function is responsible for creating a new document in Sanity for each product. It first calls the `uploadImageToSanity` function to get the image's asset ID. Then it creates a product document using the gathered data (name, description, price, image, category, etc.). If the image upload fails, the product is skipped.

## Import Products Function:

This function fetches product data from the provided external API (`https://template1-neon-nu.vercel.app/api/products`) and then loops through each product to call the `uploadProduct` function and upload it to Sanity. It performs error handling to ensure that the data fetch and upload operations are robust.

# 2. Sanity Schema File

```
import { defineType } from "sanity"

export default defineType({
    name: 'products',
    title: 'Products',
    type: 'document',
    fields: [
        {
        name: 'name',
        title: 'Name',
        type: 'string',
        },
        {
        name: 'price',
        title: 'Price',
        type: 'number',
        },
        {
        name: 'description',
        title: 'Description',
        type: 'text',
        },
        {
        name: 'image',
        title: 'Image',
        type: 'image',
        },
        {
            name:"category",
            title:"Category",
            type: 'string',
            options:{
                list:[
                    {title: 'T-Shirt', value: 'tshirt'},
                    {title: 'Short', value: 'short'},
                    {title: 'Jeans', value: 'jeans'} ,
                    {title: 'Hoddie', value: 'hoodie'} ,
                    {title: 'Shirt', value: 'shirt'} ,
                ]
            }
        },
        {
            name:"discountPercent",
            title:"Discount Percent",
            type: 'number',
        },
        {
            name:"new",
            type: 'boolean',
            title:"New",
        },
        {
            name:"colors",
            title:"Colors",
            type: 'array',
            of:[
                {type: 'string'}
            ]
        },
        {
            name:"sizes",
            title:"Sizes",
            type: 'array',
            of:[
                {type: 'string'}
            ]
        }
    ],
})
```

The schema file defines the structure of the product document in Sanity CMS. This schema determines the fields and data types for the product entries. Here is an explanation of the schema:

# A. Fields Breakdown:

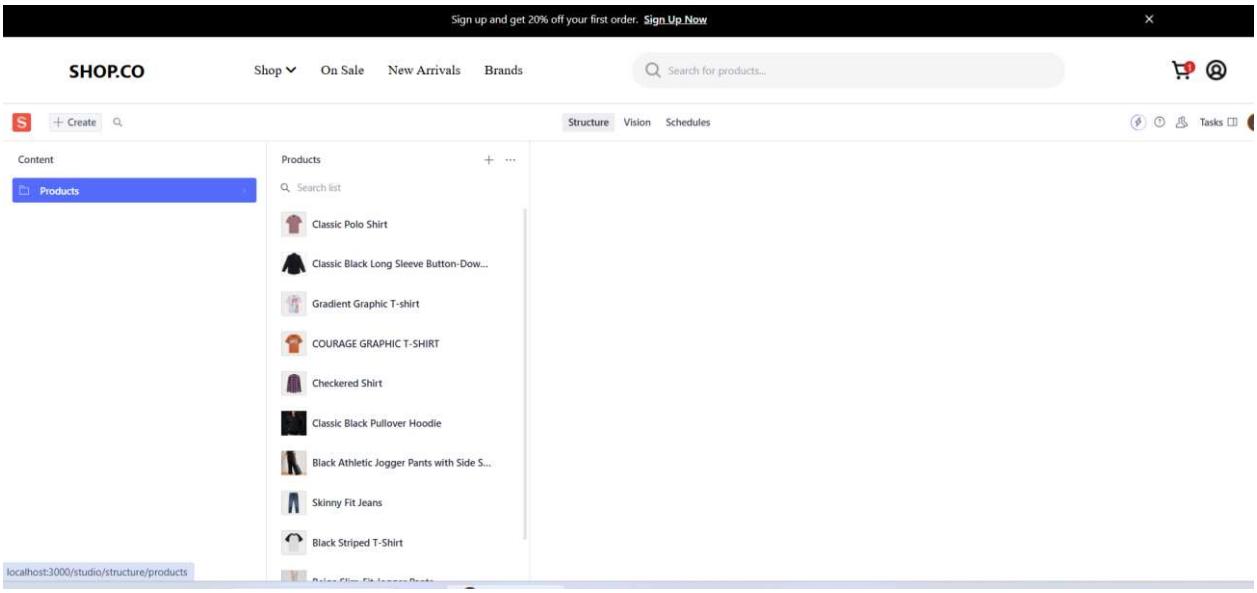
- **`name`:** A string field for the product name.
- **`price`:** A number field to store the price of the product.
- **`description`:** A text field for the product description.
- **`image`:** An image field to store the product image.
- **`category`:** A string field with options, defining the product's category (e.g., T-Shirt, Short, Jeans).
- **`discountPercent`:** A number field to store any discount on the product.
- **`new`:** A boolean field to mark the product as "new" or not.
- **`colors`:** An array of strings to hold the available colors of the product.
- **`sizes`:** An array of strings to hold the available sizes for the product.

# B. Schema Benefits:

- **Flexibility**: The schema is flexible enough to adapt to different product categories by defining various fields (e.g., colors, sizes, discount percentage).

- **Data Integrity**: Using specific data types (string, number, boolean) ensures that the data stored in Sanity is accurate and consistent.
- **Categorization**: The `category` field helps categorize products, making it easier to manage large datasets.

# 3.Successful Migration:




After successfully migrating the product data, the fields defined in the Sanity schema will be populated with the corresponding values from the external API

1. **Product Data Population in Sanity**

- **`name`:** The product name (e.g., "Red T-shirt") from the API will populate the `name` field in the Sanity document.
- **`price`:** The price (e.g., 29.99) will be stored in the `price` field as a number.

- **`description`:** The product description (e.g., "A stylish red T-shirt made from 100% cotton.") will fill the `description` field. And more ....

This data will now be structured and stored in your Sanity CMS and will be ready to be fetched and displayed in the frontend.

# 4.Displaying Data on Frontend

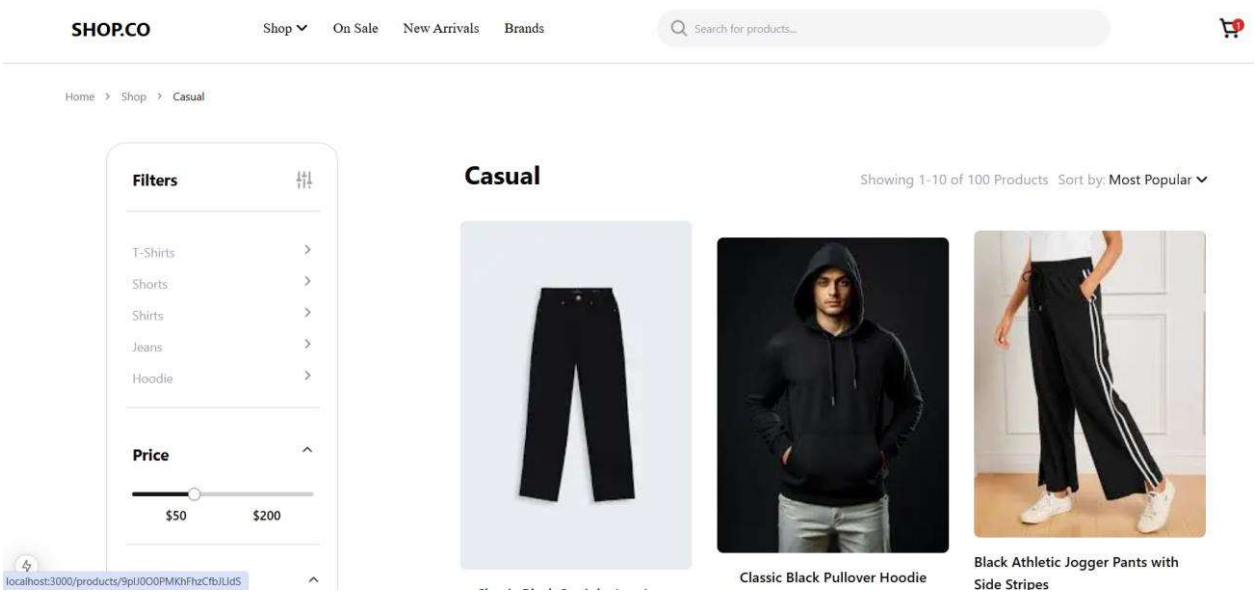
Once the data is successfully uploaded to Sanity, it can be fetched and rendered on your frontend (for example, at `localhost:3000/store`).

## Fetching Data from Sanity (in your frontend)

On your frontend application, you would use Sanity's API to query the products, typically through **GROQ** queries or **Sanity Client**.

## export const getProducts = async () => { const query = '*[_type == "products"]'; const products = await client.fetch(query); return products; };

Once the products are fetched, you can display them in your store page (`localhost:3000/store`) like so:




# Conclusion

In conclusion, the provided code and schema establish a highly effective system for migrating product data from an external API into the Sanity CMS platform. By leveraging **Sanity Client**, we are able to integrate and manipulate product data with ease. The **schema** ensures that all product information is

structured properly, making the data ready for display in a frontend marketplace.

This integration approach not only enhances the marketplace's backend functionality but also provides a seamless process for importing and organizing product data. The **data migration script** further automates the process, allowing large datasets to be migrated efficiently, reducing manual work and errors.

By utilizing this system, businesses can easily scale their marketplaces and efficiently manage product data from a variety of sources, paving the way for a more streamlined eCommerce operation.